I Erläuterungen

Voraussetzungen gemäß KCGO und Abiturerlass in der für den Abiturjahrgang geltenden Fassung

Standardbezug
Die nachfolgend ausgewiesenen prozessbezogenen Kompetenzbereiche sind für die Bearbeitung der jeweiligen Aufgabe besonders bedeutsam. Darüber hinaus können weitere, hier nicht ausgewiesene prozessbezogene Kompetenzbereiche für die Bearbeitung der Aufgabe nachrangig bedeutsam sein, zumal die Kompetenzbereiche in engem Bezug zueinander stehen. Die Operationalisierung des Standardbezugs erfolgt in Abschnitt II.

Aufgabe	Prozessbezogene Kompetenzbereiche				
	P1	P2	P3	P4	P5
1.1		X	X		
1.2	X			X	
2.1		X			X
2.2				X	
2.3			X		
3.1		X		X	
3.2			X	X	
3.3			X		
3.4			X		
4.1	X				
4.2		X		X	
4.3			X		X

Inhaltlicher Bezug
Der vorliegende Vorschlag bezieht sich schwerpunktmäßig auf die inhaltsbezogenen Kompetenzbereiche Algorithmen (I1) und Informatiksysteme (I4) nach KCGO.

Q1: Algorithmik und objektorientierte Modellierung
verbindliche Themenfelder: Such- und Sortieralgorithmen (Q1.1); Rekursion (Q1.2); Klassen und Objekte (Q1.3); Höhere Datenstrukturen und ihre objektorientierte Modellierung (Q1.4)

II Lösungshinweise und Bewertungsraster

In den nachfolgenden Lösungshinweisen sind alle wesentlichen Gesichtspunkte, die bei der Bearbeitung der einzelnen Aufgaben zu berücksichtigen sind, konkret genannt und diejenigen Lösungswege aufgezeigt, welche die Prüflinge erfahrungsgemäß einschlagen werden. Lösungswege, die von den vorgegebenen abweichen, aber als gleichwertig betrachtet werden können, sind ebenso zu akzeptieren.

Aufg.	erwartete Leistungen	BE
1.1	Zwischen den Klassen *Projekt* und *Ziel* und den Klassen *Ziel* und *Schritt* bestehen Aggregationen, weil ein Projekt aus Zielen besteht und ein Ziel aus Schritten besteht. Zwischen den Klassen *VerzeichnisAnlegen* und *Schritt* ebenso wie zwischen *DateiKompilieren* und *Schritt* besteht eine Vererbungsbeziehung, weil *DateiKompilieren* und *VerzeichnisAnlegen* spezielle Schritte sind. *Ziel* hat zu sich selbst eine rekursive Assoziation, weil jedes Ziel weitere Ziele kennt, von denen es abhängig ist. Zwischen den Klassen *Buildsystem* und *Projekt* besteht eine Assoziation, weil die Klasse *Buildsystem* die Klasse *Projekt* verwendet.	3
1.2	Das Konzept der Polymorphie in einer objektorientierten Sprache ermöglicht es, dass ein Bezeichner Objekte unterschiedlicher Klassen aufnehmen kann. So kann ein Bezeichner mit dem Typ *Schritt* deklariert werden. Dieser Bezeichner kann dann aber auch Objekte der Unterklassen von Schritt aufnehmen, z. B. ein Objekt der Klasse *DateiKompilieren* oder ein Objekt der Klasse *VerzeichnisAnlegen*. In den Klassen *VerzeichnisAnlegen* und *DateiKompilieren* ist die abstrakte Methode *ausführen()* der Oberklasse *Schritt* jeweils konkret und unterschiedlich implementiert. Wird nun auf einem Bezeichner vom Typ *Schritt* die Methode *ausführen()* aufgerufen, so ermittelt das Laufzeitsystem automatisch die korrekte Methode, je nachdem, ob der Bezeichner ein Objekt der Klasse *VerzeichnisAnlegen* oder der Klasse *DateiKompilieren* enthält.	3
2.1	1) Ein Baum darf nur eine Wurzel (Knoten, der keine eingehenden Kanten hat) haben, der Graph in Material 2 hat mit Ziel A und Ziel E aber zwei Wurzeln. 2) Jeder Knoten eines Baumes darf nur einen Elternknoten haben. Ziel G hat aber zwei Elternknoten.	2
2.2	Die Höhe eines Zieles gibt die Länge des längsten Pfades an, den man von diesem Ziel aus entlang der Kanten gehen kann. Das Ziel C hat demnach die Höhe 2, weil der längste mögliche Pfad über den Knoten „Ziel G" zum Blatt „Ziel H" führt.	2

2.3	`public boolean ermittleErreichbarkeit(){`
	`  for(int i = 0; i < abhängigVon.length; i++){`
	`    if(abhängigVon[i].erreicht == false) return false;`
	`  }`
	`  return true;`
	`}`
		4

3.1	Bevor Ziel B und Ziel E erledigt werden können, müssen die Ziele C, D, F, G, H, X und Y erledigt werden.	1
3.2	**Liste** Erster: Knoten Anzahl: int Liste() getAnzahl(): int hinzufügen(einZiel: Ziel) holeNächstes(): Ziel enthält(einZiel: Ziel): boolean **Knoten** Inhalt: Ziel Nächster: Knoten Knoten(einZiel: Ziel) getNächster(): Knoten setNächster(einKnoten: Knoten) getInhalt(): Ziel	3

Aufg.	erwartete Leistungen	BE

3.3	<pre>public boolean enthält(Ziel einZiel){
	  Knoten temp = Erster;
	  while(temp != null){
	    if(einZiel == temp.getInhalt()){
	      return true;
	    }
	    temp = temp.getNächster();
	  }
	  return false;
	}</pre>
		4

3.4	<pre>private void fügeAbhängendeZieleHinzu(Ziel einZiel, Liste Zielliste){
	  Ziel[] temp = einZiel.getAbhängigVon();
	  for(int i = 0; i<temp.length; i++){
	    fügeAbhängendeZieleHinzu(temp[i], Zielliste);
	  }
	  if(!Zielliste.enthält(einZiel)) Zielliste.hinzufügen(einZiel);
	}</pre>
		4

4.1	Die öffentliche Methode *gibErstes()* der Klasse *Prioritätswarteschlange* gibt das nächste zu bearbeitende *Ziel*-Objekt aus dem Feld *dieZiele* zurück. Sie erwartet keine Parameter. Zunächst wird in Zeile 40 überprüft, ob die Liste leer ist. In diesem Fall wird *null* zurückgegeben. Sonst wird das erste Ziel aus dem Feld *dieZiele* in der lokalen Variable *nächstesZiel* zwischengespeichert. Anschließend werden die Referenz auf das erste Element inkrementiert und die Anzahl der Elemente dekrementiert. Damit ist das erste Ziel aus dem Feld gelöscht. Zum Schluss wird das in *nächstesZiel* zwischengespeicherte Ziel zurückgegeben.	3
4.2	<table><tr><td>Index</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>Inhalt</td><td>Ziel H</td><td>Ziel D</td><td>Ziel G</td><td>Ziel F</td><td>Ziel C</td></tr></table> Erstes = 0 Letztes = 4 LetztesErreichbares = 1	3
4.3	*Mögliche Argumentation bei Entscheidung gegen die Datenstruktur verkettete Liste:* Die Elemente der Prioritätswarteschlange müssen häufig umsortiert werden. Dies lässt sich auf Feldern besonders effizient implementieren. Die Datenstruktur *Liste* hat Vorteile vor allem beim Einfügen und Entfernen von Elementen. Da im gegebenen Kontext der Warteschlange allerdings nach der Initialisierung weder Elemente hinzugefügt noch entfernt werden müssen, geht dieser Vorteil gegenüber Feldern verloren. Ein Feld eignet sich daher besser zur Verwaltung der zu erledigenden Ziele als eine verkettete Liste. *Mögliche Argumentation bei Entscheidung für die Datenstruktur verkettete Liste:* Das Initialisieren des Feldes *dieZiele* ist aufwändig und erfordert nach dem Einfügeprozess noch ein Sortieren der erreichbaren Ziele. Der Quellcode wird dadurch komplex und fehleranfällig. Bei Verwendung der Datenstruktur verkettete Liste kann jedes Ziel einfach direkt an der richtigen Position eingefügt werden, ohne dass ein Umsortieren der schon enthaltenen Elemente notwendig ist. Das reduziert die Komplexität bei der Implementierung. Eine verkettete Liste eignet sich daher besser zur Verwaltung der zu erledigenden Ziele als ein Feld.	3
	Summe	**35**

III Bewertung und Beurteilung

Die Bewertung und Beurteilung erfolgt unter Beachtung der nachfolgenden Vorgaben nach § 33 der Oberstufen- und Abiturverordnung (OAVO) in der jeweils geltenden Fassung. Bei der Bewertung und Beurteilung der sprachlichen Richtigkeit in der deutschen Sprache sind die Bestimmungen des § 9 Abs. 12 Satz 3 OAVO in Verbindung mit Anlage 9b anzuwenden.

Bei der Bewertung und Beurteilung der Übersetzungsleistung in den Fächern Latein und Altgriechisch sind die Bestimmungen des § 9 Abs. 14 OAVO in Verbindung mit Anlage 9c anzuwenden.

Der Fehlerindex ist nach Anlage 9b zu § 9 Abs. 12 OAVO zu berechnen. Für die Ermittlung der Punkte nach Anlage 9a zu § 9 Abs. 12 OAVO sowie Anlage 9c zu § 9 Abs. 14 OAVO wird jeweils der ganzzahlige nicht gerundete Prozentsatz bzw. Fehlerindex zugrunde gelegt.

Für die Bewertung in den modernen Fremdsprachen ist der „Erlass zur Bewertung und Beurteilung von schriftlichen Arbeiten in allen Grund- und Leistungskursen der neu beginnenden und fortgeführten modernen Fremdsprachen in der gymnasialen Oberstufe, dem beruflichen Gymnasium, dem Abendgymnasium und dem Hessenkolleg“ vom 7. August 2020 (ABl. S. 519) zugrunde zu legen. Demnach erfolgt die Bewertung und Beurteilung mit der Maßgabe, dass lediglich bei der Ermittlung des Prüfungsergebnisses (Note) aus Prüfungsteil 1 und 2 gerundet wird.

Darüber hinaus sind die Vorgaben der Erlasse „Hinweise zur Vorbereitung auf die schriftlichen Abiturprüfungen (Abiturerlass)“ und „Durchführungsbestimmungen zum Landesabitur“ in der für den Abiturjahrgang geltenden Fassung zu beachten.

Im Fach Informatik (Leistungskurs) werden Vorschläge zu den Themen der drei Kurshalbjahre Q1 (Algorithmik und objektorientierte Modellierung), Q2 (Datenbanken) und Q3 (Konzepte und Anwendungen der theoretischen Informatik) vorgelegt, wobei die Prüfungsleistung aus der Bearbeitung je eines Vorschlags zu jedem Halbjahresthema besteht, wofür insgesamt maximal 100 BE vergeben werden können. Ein Prüfungsergebnis von **5 Punkten (ausreichend)** setzt voraus, dass mindestens 45 % der zu vergebenden BE erreicht werden. Ein Prüfungsergebnis von **11 Punkten (gut)** setzt voraus, dass mindestens 75 % der zu vergebenden BE erreicht werden.

Gewichtung der Aufgaben und Zuordnung der Bewertungseinheiten zu den Anforderungsbereichen

Aufgabe	Bewertungseinheiten in den Anforderungsbereichen			Summe
	AFB I	AFB II	AFB III	
1	3	3		**6**
2	3	5		**8**
3	3	7	2	**12**
4	1	3	5	**9**
Summe	**10**	**18**	**7**	**35**

Die auf die Anforderungsbereiche verteilten Bewertungseinheiten innerhalb der Aufgaben sind als Richtwerte zu verstehen.